Improving The Robustness Of The Register File

A Register File Cache Architecture

Sicong Zhuang
Supervisor : Ramon Canal

Department of Computer Architecture

Universitat Politècnica De Catalunya

This dissertation is submitted for the degree of

Master of Science

September 2014

Abstract

Nowadays, the cutting-edge microprocessor fabrications are incorporating the 22-nm technology. Under current technologies and beyond the chip made within one die can posess different characteristics due to the limitation of the lithogrphy techniques. Thus, the so-called process variation has posed a great threat to the SRAM stability and yield. It is expected to worsen with the constant down-shrinking of the transistor size. Using bigger SRAM cells can mitigate this to some extent. However, a big SRAM cell can result in a lower performance and increases the overall area. In this thesis, a multi-level *register file cache* (RFC) architecture is proposed to reduce the penalty of a large register file. The results show a 3% - 11% performance drop compare to an ideal 1-cycle variation-free register file but a 3% - 10% performance gain over a more realistic 2-cycle register file. A thorough study is also conducted on how the size of the RFC. as well as the buses between the register file and the RFC affect the performance on a modern x86 processor. The overall power (Watt) and area (mm^2) of this register file architecture are evaluated as well. It is shown that the RFC incurs an area overhead of 0.2% and 0.4% the overhead of the power consumption.

In the same time, each succeeding technology generation has also introduced new obstacles to maintaining this growth rate. Transient faults due to single event upsets have emerged as a key challenge whose importance is likely to increase significantly in the next few design generations. It is a transient error which occurs during the runtime and can eventually leads to a *silent data corruption* (SDC) or system crash. The *architectural vulnerability factor* (AVF) [15] of a modern x86 microprocessor register file and of the RFC were computed in this thesis and by using a cycle-accurate architectural simulator the SEUs were injected to the RFC architecture during the execution to evaluate the performance of this architecture when face off against the *single event upset*.

Table of contents

List of figures

List of tables

Chapter 1

Introduction

1.1 Motivation

Variations in process parameters affect the digital integrated circuit (IC) and they have become a critical issue faced by the IC designers. Such variations are results of not only the fluctuations during fabrication (length, width, oxide thickness etc.) but also some aging-related variations (NBTI, PBTI, HCI etc.). The consequence is that they make the integrated circuits behave non-deterministically in terms of power consumption, stability and maximum speed degradation. This is especially critical for SRAM arrays as they are usually implemented with small transistors to increase density. As the device geometries continue to shrink and fabrication technology scales beyond 32 nm process variations are expected to be an even greater obstacle to keep up with the promise of Moore's law. [8] [9]

Meanwhile, most current superscalar microprocessors incorporate a RISC-like instruction set architecture (ISA). Such ISA implies that majority of the instruction operands reside in the register file. At the same time, high-end superscalar microprocessors that use out-of-order instruction processing have relied on the use of bigger and bigger register files to expose and exploit instruction level parallelism (ILP). A register file is essentially an SRAM array which means it falls into one of the victims of the process variations.

Another type of error, namely, *soft error* is also constantly haunting the microprocessors. An integrated circuit is suffered from *soft error* when one or more of the content bits are flipped. The nature of this type of error is temporal so it is also called *transient error*. Many sources contribute to *soft errors*, including energetic radiation particles, capacitive coupling, electromagnet IC interference, and other sources of electrical noise. The scaling of devices, operating voltages, and design margins for increasing performance and functionality raises

concerns about the susceptibility of future-generation systems to *soft errors*. Historically, such *soft errors* were mostly of concern when designing high-availability systems typically used in mission- or life-critical applications. However, because of the confluence of device and voltage scaling, and increasing system complexity, experts forecast that transient errors will be a problem in future highly integrated hardware designs.

1.2 Objectives and Contributions

The menacing process variations and soft errors will for sure affect the register file in the current and future technologies, rendering it unreliable; and consequently, bring down the performance of the entire processor.

One countermeasure is to use guard-banding by deploying sufficiently large SRAM cells in the register file in order to minimize the impact of process variation. However, this will increase the register file area and, caused by the longer wires, impact negatively the access time. Albeit several techniques are available to increase the robustness of any SRAM array (e.g. WL boosting, redundancy, ECC, etc) this thesis takes an alternate microarchitecture technique to try to remedy the performance degradation process variation and transient error introduce while incur as little overhead as possible.

The register file cache (RFC) architecture was originally proposed in the paper [6] in hope to cope with the ever-increasing delay of the multi-ported multi-cycle register file. This thesis takes one step further by utilizing the results from [6] and explores the possible bottlenecks of the architecture as well as its potential capability as a variation- and soft-error- aware architecture.

The following list is what has been done in this thesis:

- The RFC architecture is implemented in a cycle-accurate x86-family architectural simulator with the best prefetching/fetching policy combination reported in [6].
- Two vital characteristic (number of entries of the RFC and the buses in between RFC and the register file) of the RFC architecture and their impacts on the performance were analyzed in detail.
- The overall area and power of this RFC architecture is evaluated using McPAT [11].

- A strong-weak combination, where the RFC architecture is prone to the variation and results in some unusable cell blocks (decrease in size) while the register file itself is immune to the variation, was proposed to test the robustness against the process variations.
- Architecture Vulnerability Factor (AVF) [15] of the register file is computed using the architectural simulator. Furthermore, a minimalistic soft-error handling mechanism were proposed and SEUs were randomly injected during the simulation to evaluate its impact on the RFC architecture.

1.3 Organization

The rest of the thesis is composed of the following chapters. Chapter 2 provides a brief background information on the process variation and soft error. Chapter 3 describes the register file cache architecture (RFC) proposed in [6] and the prefetching/fetching policies used in this thesis. Chapter 4 evaluates and analyzes the results as well as gives information on the experiment platform and some crucial microarchitectural parameters. Chapter 5 draws the conclusion.

Chapter 2

Backgrounds

2.1 Process Variation

2.1.1 A Brief Introduction

Successful design of digital integrated circuits (ICs) has often relied on complicated optimization among various design specifications such as silicon area, speed, testability, design effort, and power dissipation. Such a traditional design approach inherently assumes that the electrical and physical properties of transistors are deterministic and hence, predictable over the device lifetime. However, with the silicon technology entering the sub 65-nm regime, transistors no longer act deterministically. Fluctuation in device dimensions due to manufacturing process (subwavelength lithography, chemical mechanical polishing, etc.) is a serious issue in nanometer technologies.

Until approximately 0.35-μm technology node, process variation was inconsequential for the IC industry. Circuits were mostly immune to minute variations because the variations were negligible compared to the nominal device sizes. However, with the growing disparity between feature size and optical wavelengths of lithographic processes at scaled dimensions (below 90 nm), the issue of parameter variation is becoming severe.

2.1.2 Classification of Variations

Variations can have two main components: *interdie* and *intradie*. Parametric variations between dies that come from different runs, lots, and wafers are categorized into interdie variations whereas variation of transistor strengths within the same die are defined as intradie variations. Fluctuations in length (L), width (W), oxide thickness (T_{ox}), flat-band

conditions, etc., give rise to interdie process variations whereas line edge roughness (LER) or random dopant fluctuations (RDFs) cause intradie random variations in process parameters. Systems that are designed without consideration to process variations fail to meet the desired timing, power, stability, and quality specifications [8].

Negative bias temperature instability (NBTI), positive bias temperature instability (PBTI), hot carrier injection (HCI), time-dependent dielectric breakdown (TDDB), and electromigration give rise to so-called *aging variation*. Temperature and voltage fluctuations fall under the category of environmental variations.

A. Spatial Variations

The spatial process variations can be attributed to several sources, namely, subwavelength lithography, RDF, LER, and so on. For the present technology regime (< 65 nm), the feature sizes of the devices are smaller than the wavelength of light, and therefore, printing the actual layout correctly is extremely difficult. This is further worsened by the processes that are carried out during fabrication. Some of the processes and corresponding sources of variations are the following:

- wafer: topography, reflectivity;
- reticle: CD error, proximity effects, defects;
- stepper: lens heating, focus, dose, lens aberrations;
- etch: power, pressure, flow rate;
- resist: thickness, refractive index;
- develop: time, temperature, rinse;
- environment: humidity, pressure.

The above manufacturing processes result in fluctuations in L; W; T_{OX} , V_{TH} (interdie variations) as well as dopant atom concentration and line edge irregularities (intradie variations).

B. Temporal Variations

Due to scaled dimensions, the operating conditions of the ICs also change circuit behavior. Voltage and temperature fluctuation manifests as unpredictable circuit speed whereas persistent stress on the devices leads to systematic performance and reliability degradation.

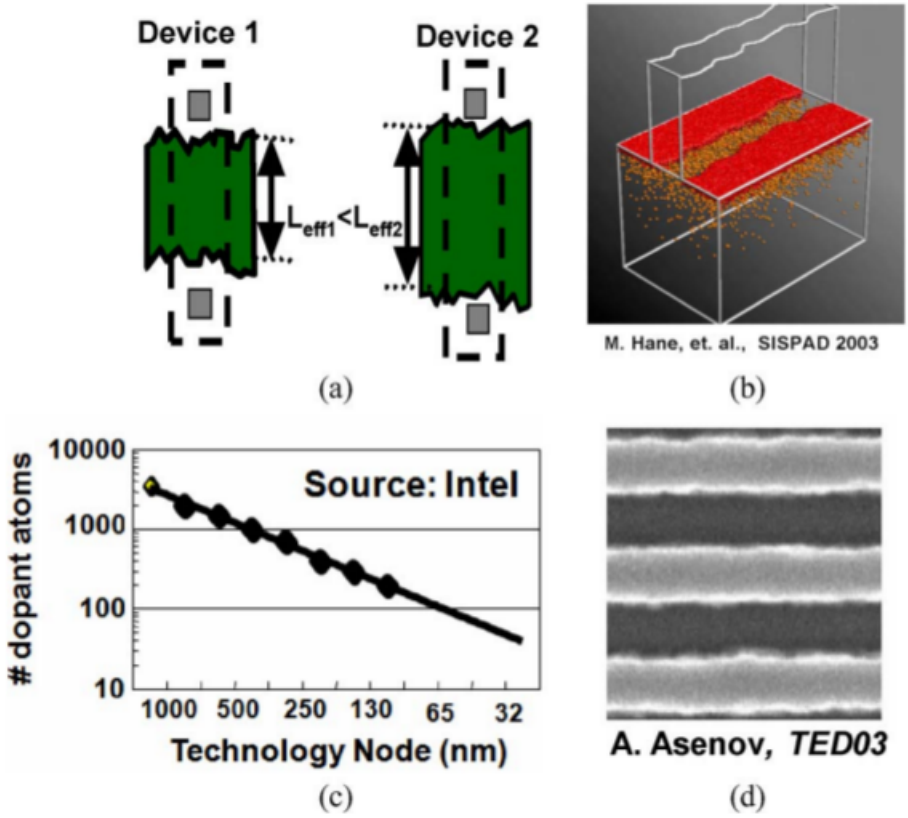

Fig. 2.1 Types of spatial variations:(a) channel length variation, (b) RDF, (c) scaling trend of number of doping atoms in the channel, (d) LER. Source: [8]

The devices degrade over a lifetime and may not retain their original specifications.

One general degradation type is defect generation in the oxide bulk or at the Si–SiO_2 interface over time. The defects can increase leakage current through the gate dielectric, change transistor metrics such as the threshold voltage, or result in the device failure due to oxide breakdown.

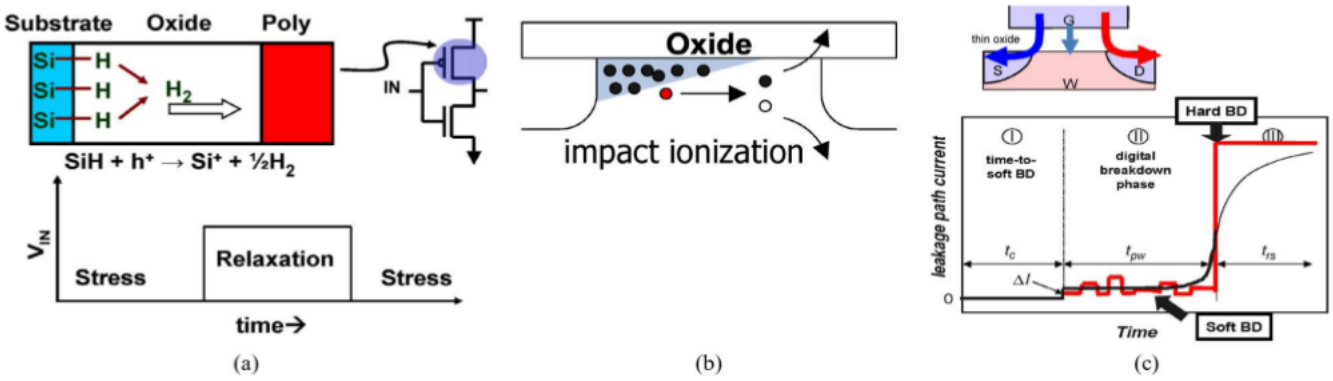

Fig. 2.2 Types of temporal variations:(a) NBTI degradation process in PMOS, (b) Impact ionization due to HCI, (c) Percolation path due to TDDB. Source: [8]

The major classification of the temporal process variations are:

- *NBTI* [2] [3] [13] is one of the most important threats to PMOS transistors in very large scale integration (VLSI) circuits. The electrical stress $V_{GS} < 0$ on the transistor generates traps at the Si–SiO_2 interface (Figure 2.2). These defect sites increase the threshold voltage, reduce channel mobility of the metal–oxide–semiconductor field-effect transistors (MOSFETs), or induce parasitic capacitances and degrade the performance. Overall, significant reduction in the performance metrics implies a shorter lifetime and poses a challenge for the IC manufacturers. One special property of NBTI is the recovery mechanism, i.e., the interface traps can be passivated partially when the stress is reduced.

- *PBTI* [22] [5] is experienced by the n-type metal oxide–semiconductor (NMOS) transistors similar to the NBTI in case of PMOS transistors. Until recently, NBTI effect for PMOS was considered to be more severe than PBTI. However, due to introduction of high-k dielectric and metal gates in sub-45-nm technologies, PBTI is becoming an equally important reliability concern.

- *HCI* [24] [1] can create defects at the Si–SiO 2 interface near the drain edge as well as in the oxide bulk. The damage is due to carrier heating in the high electric field

near the drain side of the MOSFET, resulting in impact ionization and subsequent degradation (Figure 2.2(b)). Also, HCI occurs during the low-to-high transition of the gate of an NMOS, and hence the degradation increases for high switching activity or higher frequency of operation. Furthermore, the recovery in HCI is negligible, making it worse for alternating current (AC) stress conditions.

- *TDDB* [14] [12] also known as oxide breakdown, is a source of significant reliability concern. When a sufficiently high electric field is applied across the dielectric gate of a transistor, continued degradation of the material results in the formation of conductive paths, which may shorten the anode and the cathode (Figure 2.2(c)). The short circuit between the substrate and gate electrode results in oxide failure. Once a conduction path forms, current flows through the path causing a sudden energy burst, which may cause runaway thermal heating. The result may be a soft breakdown (if the device continues to function) or hard breakdown (if the local melting of the oxide completely destroys the gate).

- *Electromigration* [23] is the transport of material caused by the gradual movement of the ions in a conductor due to the momentum transfer between conducting electrons and diffusing metal atoms. The resulting thinning of the metal lines over time increases the resistance of the wires and ultimately leads to path delay failures.

- *Voltage and temperature fluctuations* Temporal variations like voltage and temperature fluctuations add to the circuit marginalities. The higher temperature decreases the V_{TH} (good for speed) but reduces the on current V reducing the overall speed of the design. Similarly, if a circuit that is designed to operate at 1 V works at 950 mV due to voltage fluctuations, then the circuit speed would go below the specified rate. Since the voltage and the temperature depend on the operating conditions, the speed becomes unpredictable.

2.1.3 Impact on SRAM

Impact by Process Variations

The interdie parameter variations, coupled with intrinsic on-die variation in the process parameters (e.g., threshold voltage, channel length, channel width of transistors) result in the mismatches in the strength of different transistors in an SRAM cell. The device mismatches can result in the failure of SRAM cells [8]. The parametric failures in SRAM cells are principally due to:

- destructive read (i.e., flipping of the stored data in a cell while reading - known as read failure R_F);
- unsuccessful write (inability to write to a cell - defined as write failure W_F);
- an increase in the access time of the cell resulting in violation of the delay requirement - defined as access time failure A_F;
- destruction of the cell content in standby mode with the application of lower supply voltage (primarily to reduce leakage in standby mode) - known as hold failure H_F.

Another important failure model for SRAMs (especially low-voltage SRAMs)s the pseudoread problem (illustrated in Figure 2.3). When the word line is enabled to read address 0, the neighboring cells (belonging to different word) go through read disturbance. The three neighboring cells should be robust enough to sustain possible retention/weak write failures.

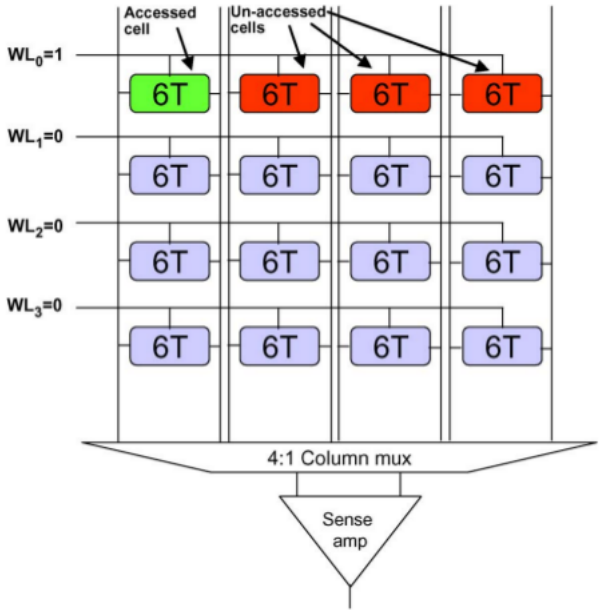

Fig. 2.3 Half select issue due to column multiplexing. The unselected words (highlighted in red) experience read disturbance when a particular word is being written. Source [8]

Impact from temporal degradation in memory

As noted previously, parametric variations in memory arrays due to local mismatches among six transistors in a cell can lead to failures. As a result, NBTI degradation, which only affects the PMOS transistors. can have a prominent effect in SRAM parametric failures. Therefore,

an SRAM array with perfect SNM may experience failures after few years due to temporal degradation of SNM. Simulation results obtained for a constant ac stress at a high temperature show that SNM of an SRAM cell can degrade by more than 9% in three years [8].

Weaker PMOS transistors (due to NBTI) can manifest itself as increased read failures. On the other hand, it also enables faster write operation, reducing statistical write failure probability. The results using Hspice are shown in Figure 2.4(b) for three different conditions: 1) only RDF, 2) static NBTI shift with RDF, and 3) statistical NBTI variation combined with the RDF. Comparison between conditions 2) and 3) shows that the impact of NBTI variation can significantly increase the read failure probability of an SRAM cell. In contrast to the read operation, write function of an SRAM cell usually benefits from the NBTI degradation. Due to NBTI, PMOS pull-up transistor PL (Figure 2.4(c)) become weaker compared to the access transistor AXL, leading to a faster write operation [8].

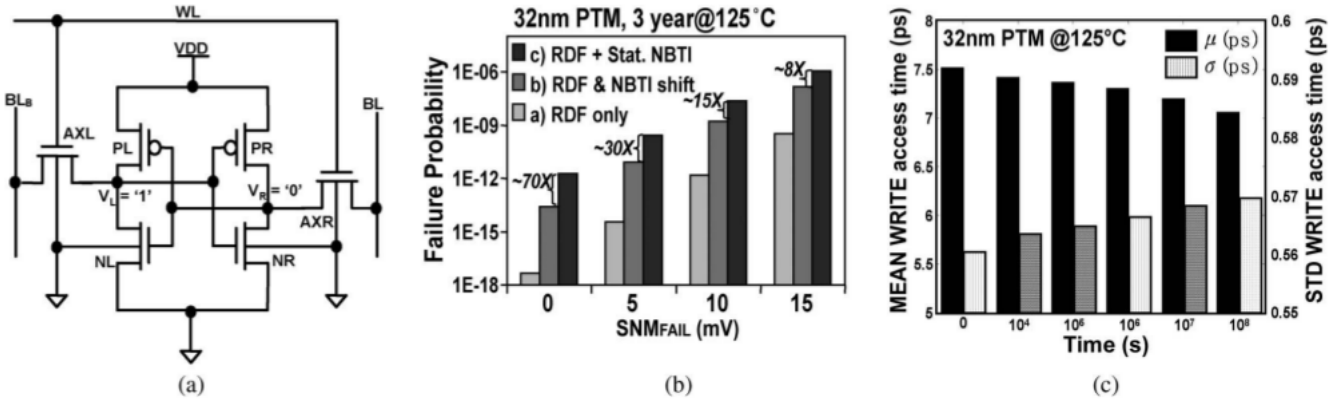

Fig. 2.4 (a) 6T SRAM bit-cell schematic. (b) SRAM read failure probability after a three-year NBTI stress. The failure probability increases due to combined RDF and NBTI. (c) Variation in write access time under temporal NBTI variation. The mean write time reduces but the spread increases. Source: [8]

2.2 Soft Error

Single Event Effects (SEEs) are induced by the interaction of an ionizing particle with electronic components. Ionizing particles can be primary (such as heavy ions in space environment or alpha particles produced by radioactive isotopes contained in the die or its packaging), or secondary (recoils) created by the nuclear interaction of a particle, like a neutron or a proton with silicon, oxygen or any other atom of the die. SEEs become possible when the

collected fraction of the charge liberated by the ionizing particle is larger than the electric charge stored on a sensitive node. A sensitive node is defined as a node in a circuit whose electrical potential can be modified by internal injection or collection of electrical charges. Usually, an information bit is associated with a voltage level [7].

Today, these effects are considered as an important challenge, which is strongly limiting the reliability and availability of electronic systems, and have motivated abundant research and development efforts in both the industry and academia. In memory devices, latches, and registers, single events are mainly called Single Event Upsets (SEU). This corresponds to a flip of the cell state. When for one particle interaction many storage cells are upset, a Multi-Cell Upset (MCU) is obtained. If more than one bit in a word is upset by a single event, a (Multi-Bit Upset (MBU) is obtained [7].

2.2.1 Single Event Upset (SEU)

Technology scaling has made today's designs much more susceptible to soft errors. The ever decreasing supply voltages and nodal capacitances (required for constraining the power and making circuit transition faster) results in reduced critical charge (Qcrit) required to upset a node in digital circuits. The problem becomes more acute for aircraft and space electronics where high-energy neutrons at higher altitudes and heavy ions in space are more abundant. Because of the prevailing predictions that soft error rate is increasing exponentially, there is a growing trend in the community to adopt soft error rate as a design parameter along with speed, area and power requirements. Furthermore, for the high-end server market, networking switches and database applications, the reliability of the computing machine is even more important along with its performance [17].

An SEU occurs when an ionizing particle strike modifies the electrical state of a storage cell, such that an error is produced when the cell is read. In an SRAM or a flip-flop, the state of the memory is reversed. In a DRAM, the charge stored can be slightly modified and interpreted as a wrong value by the read circuitry. In logic circuits, SEUs can also be obtained when a SET propagates through the combinational logic and is captured by a latch or a flip-flop [7].

Soft errors due to cosmic rays are already making an impact in industry. In 2000, Sun Microsystems acknowledged cosmic ray strikes on unprotected cache memories as the cause of random crashes at major customer sites in its flagship Enterprise server line. Sun is documented to having lost a major customer to IBM from this episode. In 1996, Normand

reported numerous incidents of cosmic ray strikes by studying the error logs of several large computer systems. The fear of cosmic ray strikes prompted Fujitsu to protect 80% of its 200,000 latches in its recent SPARC processor with some form of error detection [15].

2.2.2 Soft Error Rate (SER)

SER is the rate at which soft errors appear in a device for a given environment. When the particular environment is known or for a reference environment (NYC latitude at sea level), the SER rate can be given in FIT. In semiconductor memory, the sensitivity is often given in FIT/Mb or in FIT/device. One FIT (Failure In Time) is equal to a failure in 10^9 h.

The FIT value is either predicted by simulation or is the result of an experimental error measurement near a neutron generator representing as accurately as possible the real neutron energy spectrum. The cross-section can be used to calculate SER. If, with a total fluence of N $neutrons/cm^2$, n errors are obtained in one device, the cross-section is σ = n/N. With ϕ the particle flux in the real environment, expressed in $n/cm^2/h$, the FIT value is: FIT value = $(n/N) * \phi * 10^9$ With a cross-section of $5 * 10^{14}$ cm^2 per bit, and $\phi = 13n/cm^2/h$, the number of FIT/Mb is $5 * 10^{14} * 10^{16} * 13 * 10^9 = 650$ FIT.

Another metric vendors use to express an error budget at a reference altitude is called Mean Time Between Failures (MTBF). Errors are often further classified as undetected or detected. The former are typically referred to as *silent data corruption* (SDC); the latter, *detected unrecoverable errors* (DUE). Note that detected recoverable errors are not errors.

A group of researchers from Intel published an study on measuring SER at sea-level (Weapon Neutron Research (WNR) test facility at Los Alamos, New Mexico) [20]. They performed a comprehensive measurements and characterization of neutron-induced SER in advanced 90-nm logic CMOS technology and the neutron soft error rate dependency on voltage and area. Figure 2.6 is the wrap-up chart that shows the SRAM data as a function of the technology node. From the 0.25 μm to 90-nm node the data was measured. At the 65-nm node, the SER was obtained by extrapolation of the measured trends while assuming a supply voltage of 1.1V.

Current predictions show that typical raw FIT rate numbers for latches and SRAM cells vary between 0.001 - 0.01 FIT/bit at sea level ([15]). The FIT/bit is projected to remain in this range for the next several technology generations, unless microprocessors aggressively

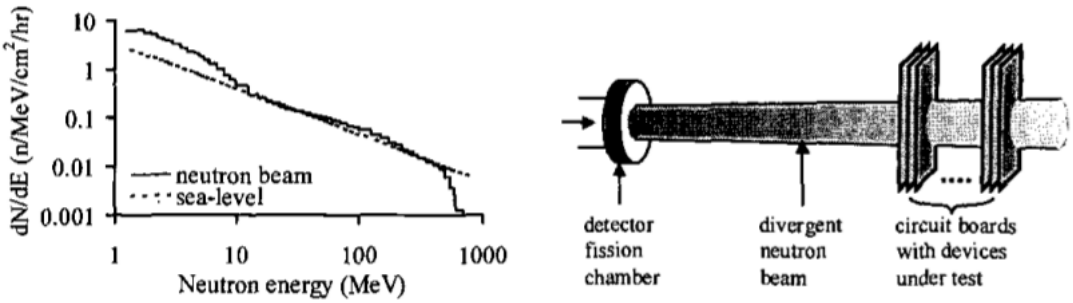

Fig. 2.5 Beam energy spectrum compared to Sea level(left); Experimental setup for SER measurements(right). Source [20]

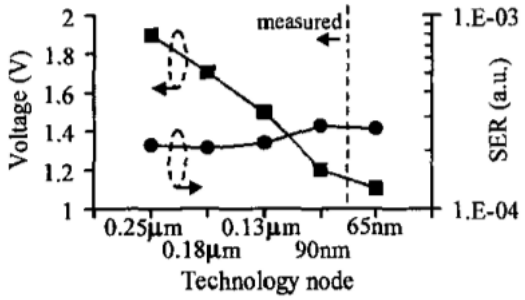

Fig. 2.6 Technology scaling of neutron SER in SRAM. Source: [20]

lower the supply voltage to reduce the overall power dissipation of chip.

Chapter 3

The Register File Cache (RFC) Architecture

3.1 Impact of A Register File

To quantify the performance hit caused by the reduction of available physical registers in the register file. The result is shown in Figure 3.1 (Register file size refers to the size of both the integer physical register file and the FP physical register file). In both benchmark suites (INT and FP) a larger register file size leads to a gain in performance. Nevertheless, to some points (somewhere between 128 and 192 entries) the performance flattens. We can observe that the performance of a register file of 192 entries has no performance difference compared to a register file of 256 in both benchmark suites. This leads to an ideal size of a register file size in a modern out-of-order x86 microprocessor (see Table 4.1 for more configuration details) in terms of performance, namely, somewhere around 192 and 256. A register file of such size with 8 read ports and 4 write ports can hardly achieve a one cycle access time and be 100% process variation resistant.

The need of a large register file size of a modern out-of-order processor along with the requirement to implement a register file using large SRAM cells to minimize the impact of process variations leads to an overprovisioned register file. The register file provides the source operands and stores the results of most instructions. The register renaming mechanism of a dynamically scheduled processors rename logical registers to physical registers at run time such that each result produced by any instruction in-flight is allocated to a different physical register. Under this scenario, the register file access time could be critical.

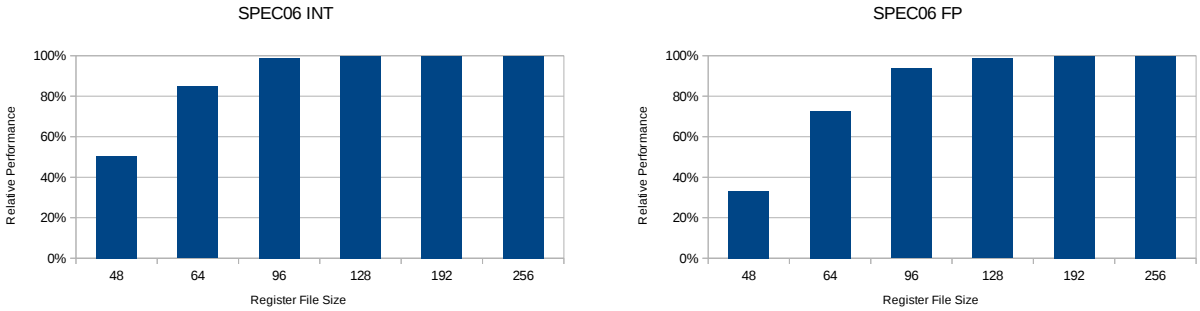

Fig. 3.1 Performance with various register file size

Compare to an ideal 1-cycle register file, a register file with 2-cycle access time 1) resolves branches one cycle later 2) requires an extra level of the bypass network which increases the complexity significantly.

At the expense of a lower performance, the complexity of the bypass network of a 2-cycle register file can be reduced by only implementing one level of the bypass network. Figure 3.2 shows the performance comparison between an ideal 1-cycle register file and a 2-cycle register file in which only one level of bypass is implemented . We chose to keep the bypass network from the last level (the second cycle of the 2-cycle writeback stage) because otherwise there would be an undesirable situation in which a value is available in one cycle via the bypass network and becomes inaccessible in the subsequent cycle then can be accessed via the register file in the next cycle. This would greatly increase the complexity of the issue logic.

In Figure 3.2, it can be observed an average performance drop of around 13% for a 2-cycle 1-bypass-level register file compared to an ideal 1-cycle register file (harmonic mean is used) in both SPEC06 INT and SPEC06 FP benchmark suites. This is quite a significant performance lose due to the larger register file access time and the lack of the full bypass network implementation.

3.2 The RFC Architecture

While originally proposed to cope with big (and slow) register files, we reevaluate the register file cache [6] as a mechanism to tolerate and hide the effect of process variations in the register file. The original proposal consists of a two-level cache-like register file architec-

Fig. 3.2 Performance drop of a 2-cycle 1-bypass level register file compare to an ideal 1-cycle register file

ture. Figure 3.3 is an illustration of the aforementioned architecture. It is a heterogeneous design in the sense that there is a register file bank at each level and the two register file banks vary in terms of entries which in turn leads to a different access time.

As shown in Figure 3.3, the levels in the register file cache are named as uppermost level and lowest level. The register file bank at the uppermost level serves as the "cache" in this architecture. Hence its name, register file cache. It provides register values required by the instructions on-the-fly and stores register values which are predicted to be used in the near future or expected to be used again. Therefore the register file bank at this level should possess as many ports as a conventional monolithic register file but fewer entries. On the other hand, the register file bank at the lowest level functions as the pool of physical registers. It stores all the physical registers (either freed or renamed) including those reside in the register file cache and all the results are always written back to this level. This implies a large number of entries. As of the ports between the register file (lowest level) and the register file cache (uppermost level) will be studied in Chapter 4.1.

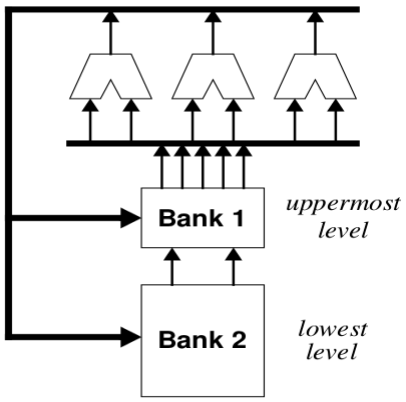

Fig. 3.3 A two-level Register File Cache architecture. Source [6]

Since there is a presence of a cache-like organization, there is a need to determine which values are cached and to predict whether to prefetch a certain value along with the replacement policy. Two caching policies were mentioned based on the authors observation of the fact that register values are often used only once [6]. The first of which is called *non-bypass* caching policy. In this caching policy only those results not read from the bypass network are cached in the register file cache (uppermost level) whereas the values which are bypassed are written back only to the register file (lowest level). Another caching policy is *ready* caching which we only cache the results that are a source operand of an instruction not yet issued but has all its other source operands ready.

Orthogonal to the caching policies two fetching mechanisms were also mentioned. In *fetch-on-demand* register values are fetched to the register file cache (uppermost level) when they are source operands of some instructions which have all their other source operands ready but those reside in the register file (lowest level). *Prefetch-first-pair* exploits the fact that the rename and issue logic of a conventional processor is able to identify all the operand communications between the instructions in the issue-window. Whenever an instruction is issued, the issue queue will be scanned to find the first instruction that needs the destination operand of the current instruction as one of its source operands. The other operands of the same instruction will then be prefetched into the register file cache (uppermost level). An example is showing below:

p1 = p2 + p3;
p4 = p3 + p6;
p7 = p7 + p8;

The aforementioned architecture forms the base of our work in this thesis since we are expecting a 2-cycle register file to reach a performance as close as that of an ideal 1-cycle register file. However, because here we are tackling with the process variation problem we are going to assume a weak-strong pair. This is to say that even a 2-cycle register file cannot be free of the process variations. Over time some of the cell of the register file will simply become unreachable which effectively reduce the capacity of the register file, hence “weak”. On the other hand, since the register file cache is relatively simple we can make it free of the process variation and becomes the “strong” part of this pair. Later we will see how well can the register file cache architecture dealing with the performance drop brought along by the weak-strong configuration.

Chapter 4

Experiments and Evaluations

4.1 Experiment Platform

The performance was evaluated using a cycle-accurate microarchitectural simulator: MARSSx86 [18]. It simulates a modern single-/multi-core x86 processor (both in-order and out-of-order). The out-of-order mode (the one used in this thesis) incorporates an issue queue, a RAT (Register Aliasing Table), physical register files for both integer and floating point registers. It implements two ROBs (Re-Order Buffer), a speculative ROB which resides at the renaming stage and another ROB which resides at the commit stage and is used to recover from branch mispredictions and interrupts. Figure 4.1 illustrates the execution flow of MARSSx86 whereas Figure 4.2 depicts the pipeline stages of the out-of-order core model used in the simulation.

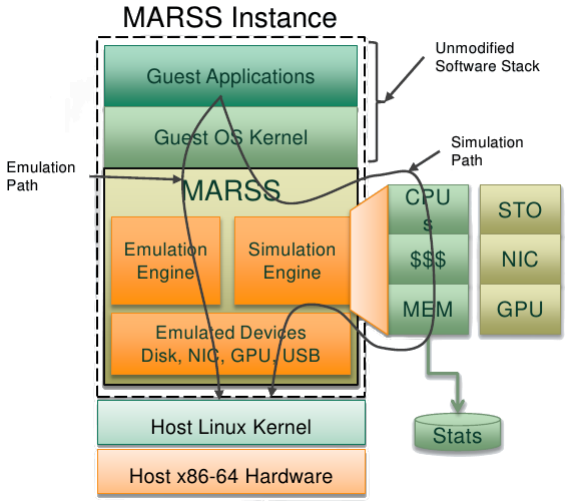

Fig. 4.1 The block diagram of the Marssx86 structure. Source [18]

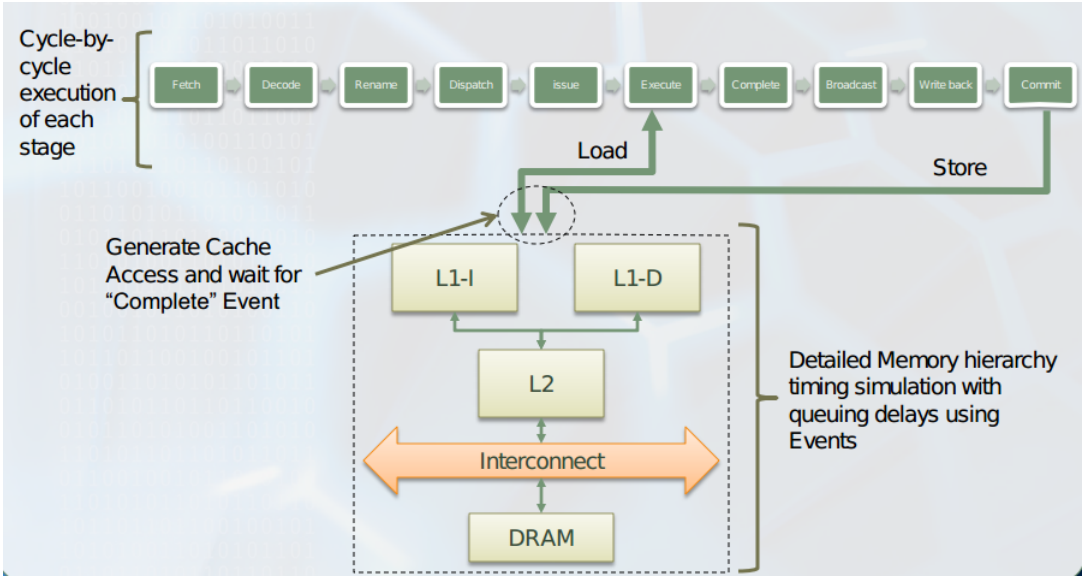

Fig. 4.2 Pipeline stages of the out-of-order core model. Source [18]

It models a 6-stage pipeline (instruction fetch; decode; rename; issue; execute; write-back; commit). The source operands are read in the same stage as it is issued. All the stages takes exactly one cycle except for execute stage which can take several cycles depends on the instructions. We modified the relevant parts of the simulator in order to properly simulate the register file cache architecture. Table 4.1 lists the major microarchitecture features we used throughout all the experiments. We implemented the register file cache (uppermost level) as a fully-associative organization with an LRU replacement policy. As of the caching policy and fetching mechanism we chose *non-bypass* caching and *prefetch-first-pair* fetching because this combination achieved the highest IPC according to [6].

For the evaluation of the register file cache architecture, we ran the SPEC CPU 2006 benchmarks [4] on the MARSSx86 simulator. SPEC CPU 2006 is the latest version of a series of benchmarks designed to provide a comparative measure of compute-intensive performance across the widest practical range of hardware using workloads developed from real user applications. It contains two benchmark suites: INT which is composed of 12 benchmarks and measures the compute-intensive integer performance; FP which is composed of 17 benchmarks and measures the compute-intensive floating point performance [10]. All the results presented in this thesis are based upon the IPC (instruction per cycle) and other relevant hardware counter data from the execution of the benchmarks.

Parameter	Value
μops operands	1 destination operand and up to 3 source operands (source operand can be either register or immediate value)
Fetch width	4 instructions
I-cache	128 KB, 8 way-associative, 64 byte lines, 2-cycle hit time, 5-cycle miss time
Branch predictor	Combined (Gshare with 2-bit bi-modal)
Reorder buffer size	128
Issue queue size	64
Functional units	4 int, 4 FP, 4 load/store
Load/store queue	64
Issue mechanism	4-way out-of-order issue
Physical registers	256 int /256 FP
D-cache	128 KB, 8 way-associative, 64 byte lines, 2-cycle hit time, 5-cycle miss time
Commit width	4 instructions
Register file access time	2 cycles
Register file cache access time	1 cycle

Table 4.1 Microarchitecture parameters used for simulation

Since the SPEC CPU 2006 benchmarks have significantly larger dynamic instruction counts and data footprint than the earlier SPEC CPU 2000 benchmarks [16], a full execution of the benchmarks on the simulator would take days even weeks to complete. In order to reduce the execution time while retain the accuracy of the simulation result we used some profiling and instrumentation techniques. We used Intel®PinPoints [19] to analyse and dynamically instrument the benchmarks. Consequently, it generates a BBV (basic block vector) file of the instrumented benchmark which we then fed it to SimPoint [21]. Given the instruction slice (number of instructions) SimPoint will in turn pick up the most representative instruction slices and provide the weight of each of them. In this thesis the instruction slice we used is 250 million which is reported to be relatively accurate [16].

4.2 Impact of Process Variation

The constant presence of process variations renders the register file itself unstable. Cells of the register file could be temporarily malfunctioning or even broken over time thus reduces the register file size temporarily or permanently. Figure 4.3 illustrates the impact of the

degradation of the register file on the performance. Here we assume a weak-strong pair architecture, the register file size degrades whereas the register file cache retains its original size. Also we introduce two baselines, a best-case scenario (an ideal 1-cycle monolithic register file) and a worst-case scenario (a 2-cycle monolithic register file with only one level of bypass network) . We used four different sizes of the register file cache, we will devote the next section analysing their impacts on the performance.

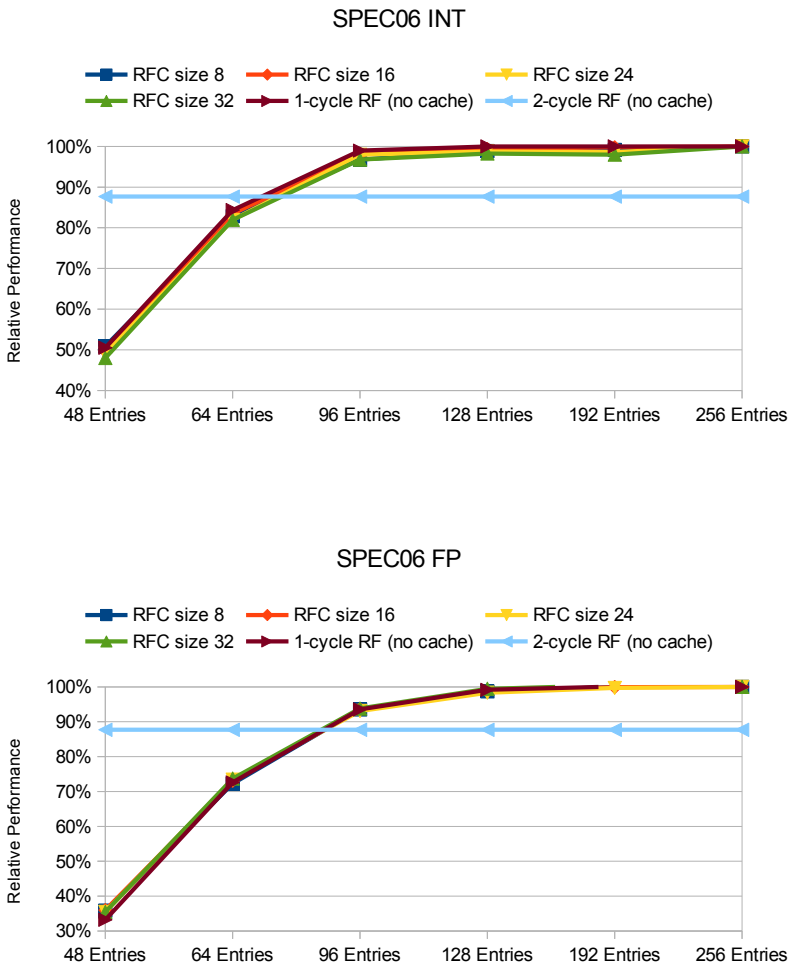

Fig. 4.3 We assume a strong register file cache with a weak register file

Not surprisingly, we observe a performance drop whenever the register file size decreases in both SPEC CPU 06 INT and FP benchmark suites. The performance degradation is within 5% after the register file size decreases to 96 entries in the INT benchmark suite. For FP the degradation is within 9%. As the register file size continues to decrease the per-

formance curves become steeper because the simulator incorporates 80 physical registers, as the size of the register file is approaching down to or even below this number the register file would be getting difficult to serve all the requests from the issue queue on time.

We also observe that despite some minor fluctuations, among the four sizes of the register file cache the performance curves see a rather similar pattern. In the INT benchmark suite the register file cache architecture sees a greater degradation (2% - 3%) of performance than an ideal 1-cycle monolithic register file when the register file size is somewhere around 64 to 96 entries. However, the performance curves converge when the register file size keeps increasing or decreasing. Whereas in FP benchmark we do not see this pattern, all the curves follow a quite close if not identical pattern.

4.3 Impact of the RFC size

Similar to a conventional cache memory system the size of a register file cache is critical to achieving high performance. Figure 4.4 shows the performances of various register file cache sizes (8, 16, 24, 32 entries) compare to an ideal 1-cycle monolithic register file. We introduce another parameter here: the buses between the register file and the register file cache which we will discuss in the next section. Regardless of the number of buses in-between these two register file banks, there is a performance gain as the register file cache size increases. However, as we can observe in the Figure 4.4 the trend flattens when the register file cache size approaches 32 entries which implies a performance bottleneck of the register file cache architecture.

Overall the register file cache architecture incurs performance lose of around 3% - 10% in SPEC CPU 2006 INT benchmark suite depending on the register file cache size compare to an ideal 1-cycle register file as well as the buses in-between. Nevertheless, it still possesses a 3% - 10% speed-up over a 2-cycle register file with one bypass level in both benchmark suites.

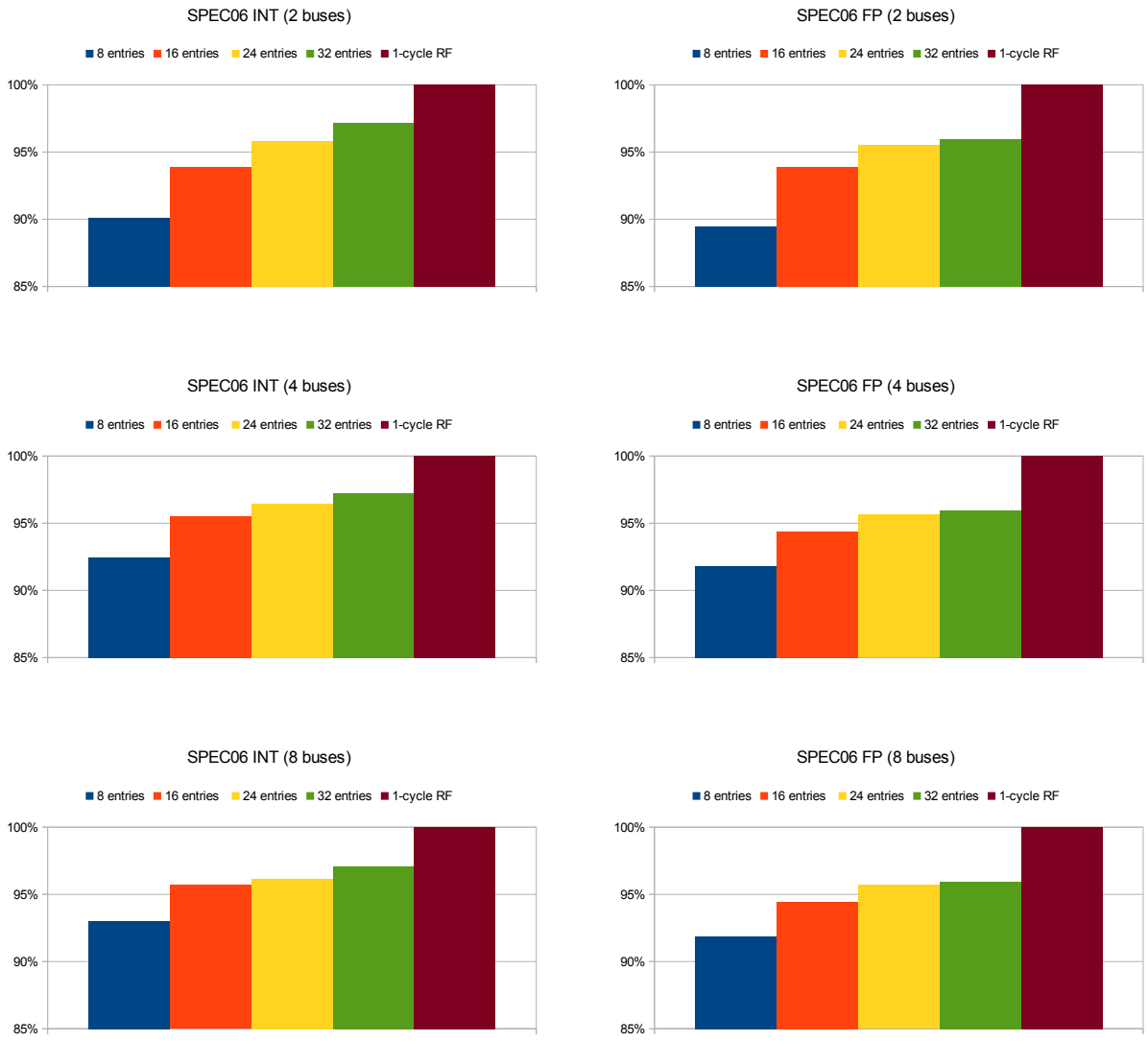

Fig. 4.4 Performance of various register file cache sizes

4.4 Usage of The Buses

In the previous section, we have seen that regardless the number of available buses between the register file and the register file cache the performance trends are nearly identical. This leads us to a further check. The results show in Figure 4.5 and 4.6. Obviously the performance of deploying 4 and 8 buses are nearly identical regardless the register file cache size while with 2 buses the performance sees a minor drop with the register file cache size of 8 and 16. However, a check on the actual statistic reveals that the performance drop is not significant, less than 2%. Hence, of all the register file cache size we have experimented the impact of using either 2, 4 or 8 buses is not apparent which implies the rare usage of the buses.

Fig. 4.5 Impact on performance of the number of buses between the RF and the RF Cache

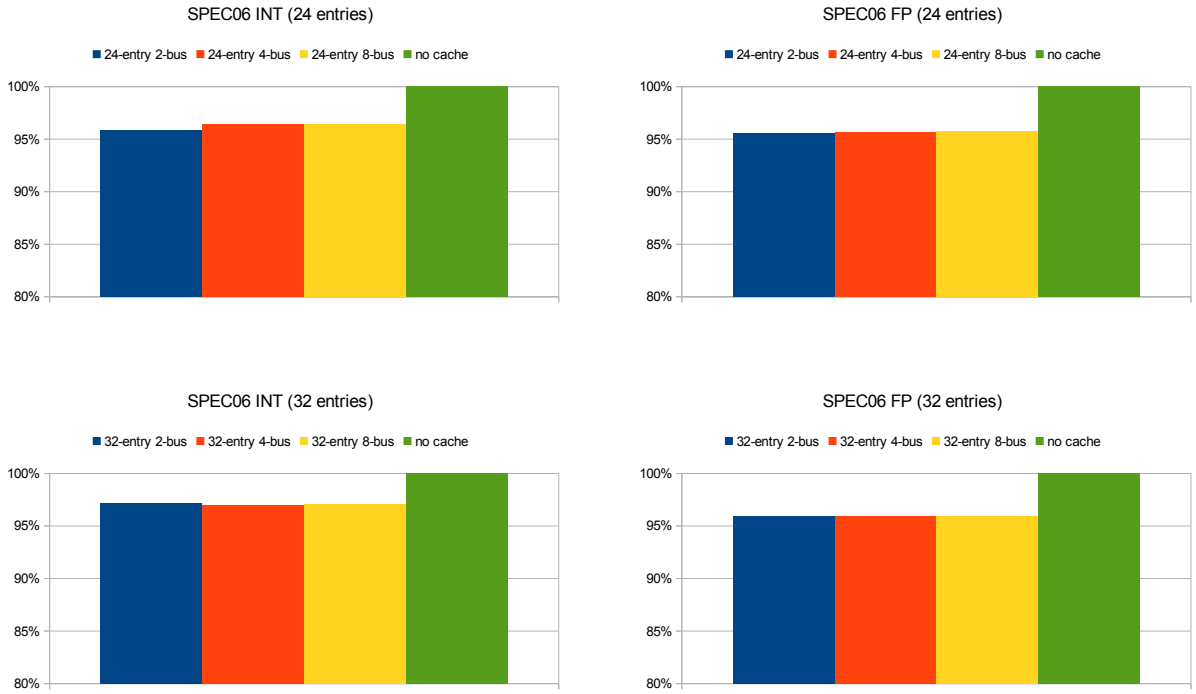

Fig. 4.6 The buses in-between have minor impact on performance (continue.)

We gathered the bus usage information in every cycle during the simulation (shown in Figure 4.7). Due to the presence of the prefetching mechanism on top of the fetch-on-demand fetching policy, 2 buses are barely enough to meet the read requests to the register file cache. It is especially prominent for small register file caches. We can see in Figure 4.7 that with a register file cache size of 8 and 2 buses these 2 buses are quite busy for more than 60% of the cycles in the SPEC CPU 2006 INT benchmark suite. Naturally, the bus demands reduces with the increasing of the register file cache size since it is able to hold more registers at a time. When 4 or 8 buses are incorporated the cycles where there is no traffic becoming more and more prominent up to close to 50% for the SPEC CPU 2006 INT benchmark suite and around 60% for the SPEC CPU 2006 FP benchmark suite . This somehow explains why in these 2 bus configurations the number of buses does not matter any more.

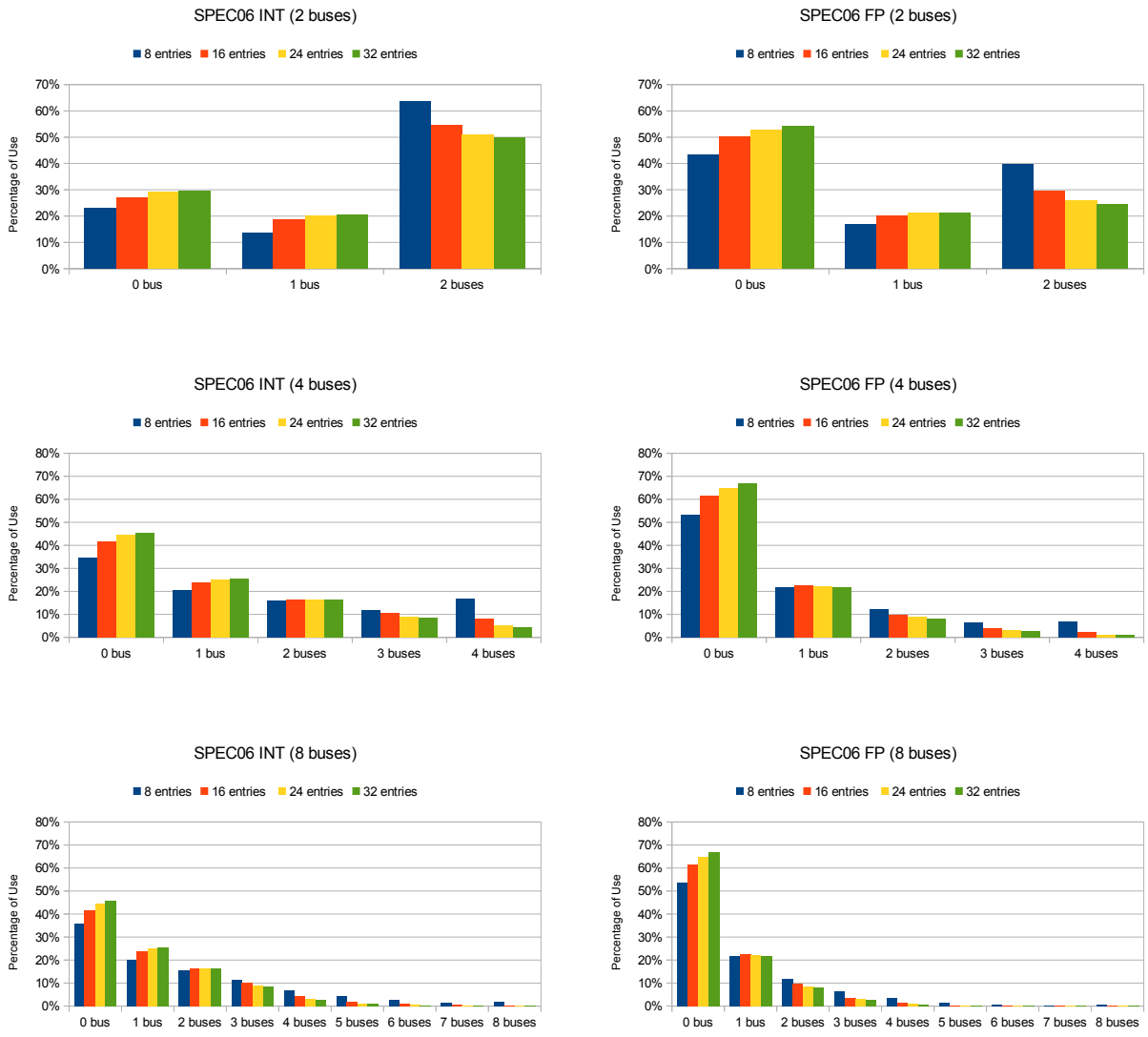

Fig. 4.7 Bus usage with various bus numbers

4.5 Usage of The Bypass Network

As we have seen in the last section, the bus traffic between the register file and the register file cache does not seem to be heavy, two buses is sufficient to achieve a relatively high performance with the maximum performance drop of less than 2%. This could imply a frequent use of the bypass network. The percentage of use of the bypass network under various number of buses is shown in Figure 4.8. First, we can observe that the number of buses has minor effect on the percentage of the usage of the bypass network. In SPEC06 INT benchmark suite of all the execution cycles around 50% of which the bypass network is not used. This number decrements when the register file cache size becomes larger. On the other hand, a one-bypass usage takes place in around 34% of all the execution cycles, it slightly increments corresponding to the increase of the register file cache size. The two-bypass and three-bypass usage, which occur less frequently, are 12% and 2% respectively.

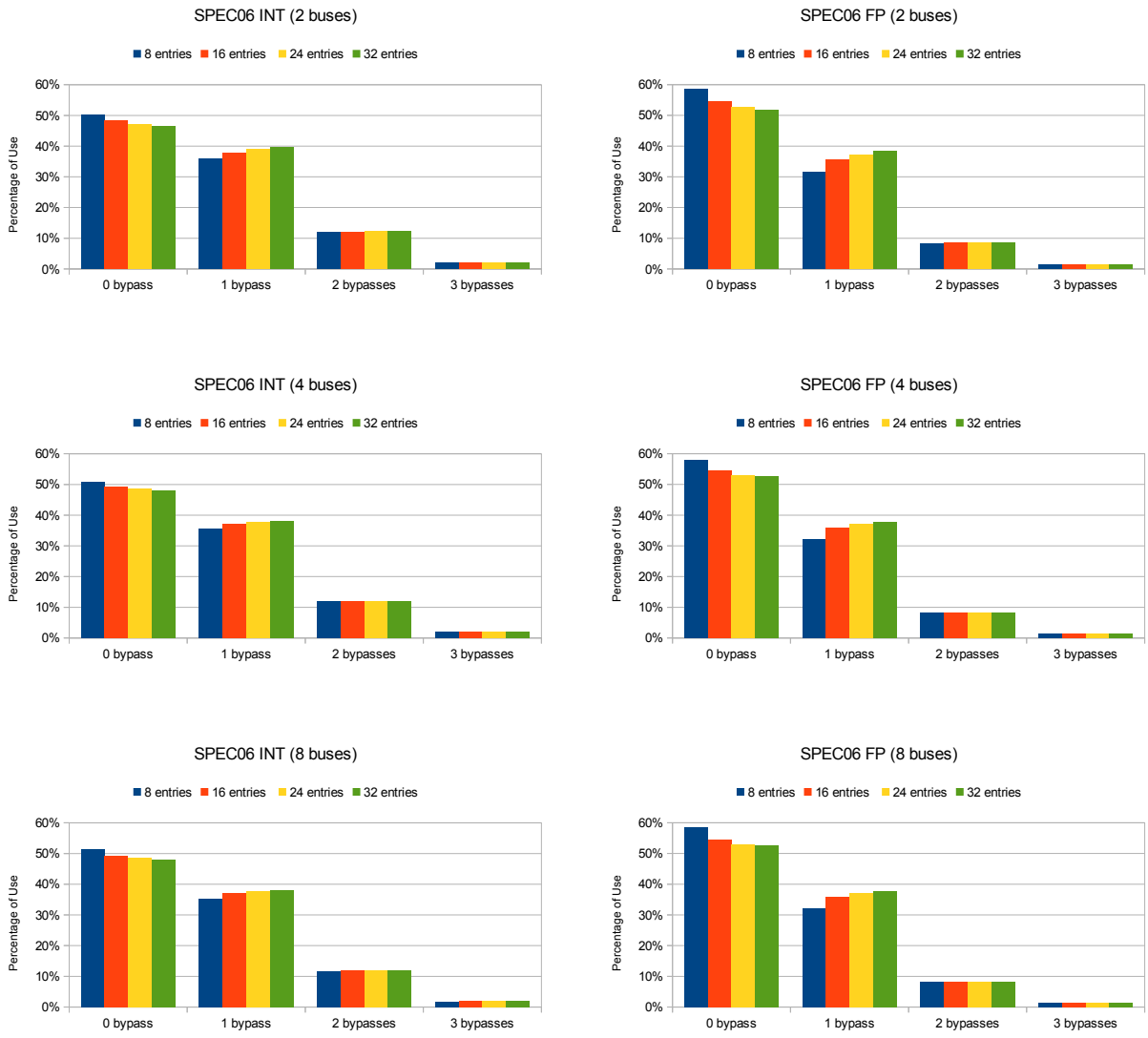

Fig. 4.8 Bypass usage with various bus numbers

Given the frequent bypass network traffic, we set out to study its relationship with the μops being executed. We distinguish the μops by the number of source operands they have since the number of the source operands has direct impact on the bypass traffic. As in Table 4.1, each μops in can have up to 3 source operands, we thus divide the μops into 4 categories. Figure 4.9 shows the harmonic mean of the percentage of the μops of each category from both the SPEC06 INT and FP benchmark suites. We can observe that the μops with 2 source operands are dominant (69% in SPEC06 INT and 67.5% in SPEC06 FP) compare to others. In Table 4.2 we list some typical μops from each type of the category in hope for a better understanding of Figure 4.9:

From Table 4.2, we can see that most of the commonly used μops has 2 source operands which explains the dominantly high percentage of 2-source-operand μops from Figure 4.9.

Category	**Typical μops**
μops with no source operand	unconditional branch (*bru*), *nop*
μops with 1 source operand	conditional branch (*br.cc*), indirect jump (*jmp*, *jmpp*), FP convertion (*cvtf*)
μops with 2 source operands	*mov*, logical (*and*, *xor* etc.), integer/FP arithmetic (*add*, *mul*, *addf*, *mulf* etc.), load (*ld* etc.), shift (*shl*, *shr*), rotate (*rotl*, *rotr*), *mask*, bit testing (*bt*, *btr*, etc.), *cmpf*
μops with 3 source operands	store (*st* etc.), conditional set/select (*set.cc*, *sel.cc*), conditional compare and set (*set.sub.cc*, *set.and.cc*), *collcc*

Table 4.2 Typical μops classified by the number of operands

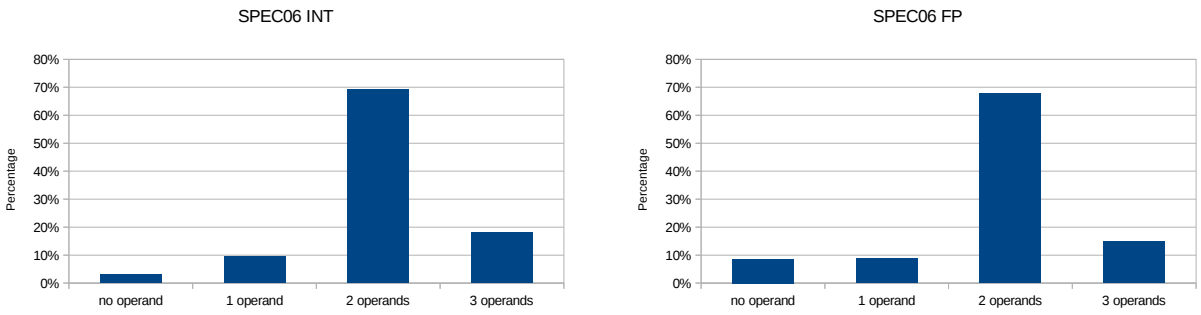

Fig. 4.9 Distribution of μops with different numbers of source operands

We also studied how do 2-source-operand μops get their source operand values (either from the register file cache or from the bypass network) and the result is shown in Figure

4.10. As indicated in Figure 4.6 the buses has minor effect on the performance plus the statistic of different buses configurations does not differ much, we hence use the smallest number of buses possible (2 buses) since the buses can significantly increase the overall area and power. In the SPEC06 INT benchmark suite, the situation that the μops read both their source operand values from the register file is around 45% and this figure decrements as the register file cache size increases. In the SPEC06 FP the percentage is 6% higher. This is not particularly surprising, given that the majority of the μops has 2 source operands, the result in Figure 4.10 should reflect the overall trends shown in Figure 4.8.

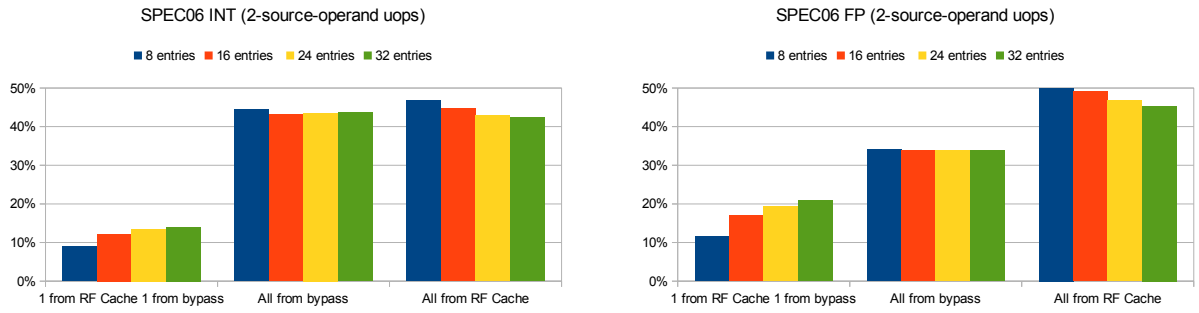

Fig. 4.10 The distribution of sources where 2-source-operand μops get their source operand values

We also would like to know the overall performance of the issue queue under the influence of the register file cache architecture by finding out the percentage of μops failed being issued due to the absence of their operands values in the register file cache. However, there are several factors that can prevent a given μop from being issued (operand not ready, no available function units, already reach the issue width etc.). In order to rule out the rest of the factors, we modified the simulator to gather the following information: the number of μops which are within the issue width and meet the *fetch-on-demand* policy but do not have all the values of their operands available in the register file cache. The result is presented in Figure 4.11. We can see that in the SPEC06 INT benchmark suite for a register file cache size of 8 entries, the percentage is around 26% and this number decreases when the capacity of the register file cache increments. Since the register file cache is able to hold more and more values, this becomes obvious. The trend of this flattens which indicates a possible bottleneck of the register file cache size. Similar percentage and tendency can also be observed in the SPEC06 FP benchmark suite.

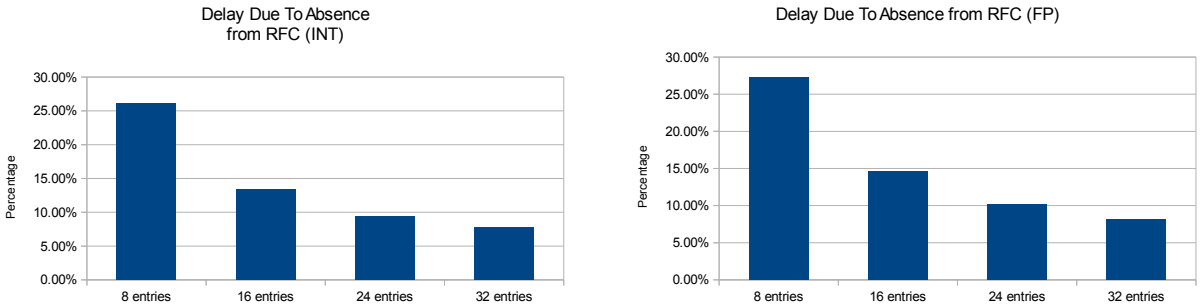

Fig. 4.11 The percentage of μops that could not be issued due to the absence of their operands values in the register file cache

4.6 Power and Area Evaluation

Apart from studying the performance impact of the register file cache, we evaluated its impact on the power consumption and the area overhead as well. We chose the McPAT [11] from HP Labs as the evaluation framework. McPAT is an integrated power, area and timing modelling framework for multi-threaded, multi-core/many-core processors. It provides a set of complete hierarchical models from architecture to the technology level by the means of an XML-based interface.

Since the process variation has become and is predicted to be one of the greatest threaten to integrated circuits, we chose the model of Intel Xeon processor of 22 nm technology during the evaluation which is the most recent technology node available in the current version of McPAT. Marssx86 has support to McPAT by the means of properly dumping the simulation results to McPAT's XML interface. The only thing we modified is the register file read and writing statistics. The lowest-level register file is read when the data from it is brought to the register file cache whereas every destination operand value is written back to it in the writeback stage. On the other hand, every read request of the operations from the issue queue is a read operation to the register file cache and the write operation to it is a combination of both the data transfer between the register file and the register file cache and the destination operand values in the writeback stage that are not read by the bypass network.

Figure 4.12 is a comparison of both the overall processor area (mm^2) and the area of the register file among 4 different register file cache sizes and an ideal 1-cycle register file as well as a 2-cycle 1-bypass register file. Since we are adding up components on top of the original processor it is reasonable to expect an overhead in terms of the area and the larger the register file cache size is the larger the area overhead is. We assume that to make the

register file more reliable the transistor that the 2-cycle 1-bypass register file made of would be slightly larger. We suppose it would take 30% more space than that of an ideal 1-cycle register file. We can see from Figure 4.12 that the register file caches still take less area than a 2-cycle 1-bypass register file.

A register file cache architecture implies more transistors over a conventional register file and it inevitably increases the power consumption. Our result shows that the register file cache architecture incurs an overhead of 0.4% and generally the overhead is positively correlated to the register file cache size.

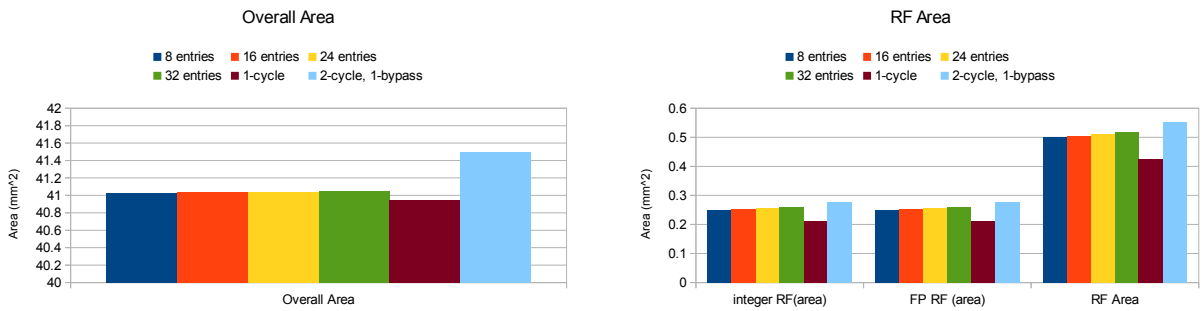

Fig. 4.12 Area of the processor and register file with/without register file cache

We further made an energy delay square metric diagram shown in which the numbers are relative to the result from the 2-cycle 1-bypass register file (Figure 4.13). It shows that in terms of energy efficiency the register file cache size of 8, 16, 24, 32 possess a rather similar efficiency (around 72% - 79% compare to the 2-cycle 1-bypass register file). Without any doubt the ideal 1-cycle register file achieves is 33% more energy-efficient than a 2-cycle 1-bypass register file.

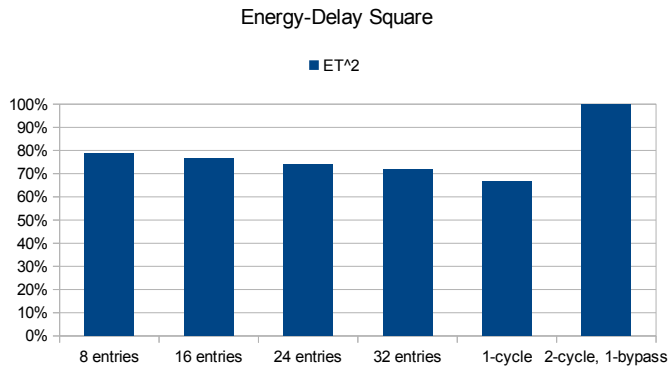

Fig. 4.13 The Energy-Delay Square metrics

4.7 Architectural Vulnerability Factor (AVF) of the RFC

4.7.1 Architectural Vulnerability Factor (AVF)

In [15], the authors recognize the soft error as a major challenge for the microprocessor designers. They listed some of the existing techniques(special radiation-hardened circuit designs, localized error detection and correction, architectural redundancy, etc.) in hope to reduce the impact of the soft error. Nevertheless, they argue, that all of these approaches introduce a significant penalty in performance, power, die size, and design time. Although a microprocessor with inadequate protection from transient faults may prove useless due to its unreliability, excessive protection may make the resulting product uncompetitive in cost and/or performance. Plus, tools and techniques to estimate processor transient error rates are not yet available or mature enough.

Instead, they proposed to use the probability that a fault in a processor structure will result in a visible error in the final output of a program that structure's *architectural vulnerability factor* (AVF). They reasoned that the key to generating these error-rate estimates is understanding that not all faults in a microarchitectural structure affect the final outcome of a program. As a result, an estimate based only on raw device fault rates will be pessimistic, leading architects to over-design their processor's fault-handling features [15].

To give an example they argued that the branch predictor's AVF is 0%. In contrast, a single-bit fault in the committed program counter will cause the wrong instructions to be executed, almost certainly affecting the program's result. Hence, the AVF for the committed pro-

gram counter is effectively 100% [15]. These are two extreme cases, it is obvious that the AVF for many other microarchitectural components will be some where between these two.

They think that by incorporating AVF into the early-stage design a processor architect can map the raw fault rate (dictated by process and circuit issues) to an overall processor error rate, and thus determine whether the design meets its error rate goals (set according to the target market). Significantly, this allows an architect to examine the relative contributions of various structures and identify the most cost-effective areas to employ fault protection techniques.

4.7.2 AVF for the RFC

The way the authors in [15] propose to compute the AVF for one microarchitectural component is to track the processor state bits required for *architectural correct execution* (ACE). Any fault in a storage cell that contains one of these bits, which we call ACE bits, will cause a visible error in the final output of a program in the absence of error correction techniques. As can be easily deduced, a fault on an un-ACE bit will not incur an error in the execution.

Identify the ACE bits

For the sake of a general discussion, the authors in [15] listed various criteria on identifying an ACE bit in a microarchitectural component. However, the scope of the thesis focuses on the register file and its attached cache-like architecture. Identifying the ACE bits becomes a rather straightforward task.

As register file provides the actual data to each and every instruction in the pipeline, in this thesis we consider all of them critical to the correct execution. Nevertheless, the value of a register only becomes valid when the physical register is assigned to an architectural register. We take that into account and only consider valid registers in the following experiments. As an attachment to the register file, the methodology also apply to the RFC.

Computing AVF

The AVF of a storage cell is the fraction of time an upset in that cell will cause a visible error in the final output of a program. Thus, the AVF for an unprotected storage cell is the percentage of time the cell contains an ACE bit. To extend it to an entire hardware structure: The AVF for a hardware structure is simply the average AVF for all its bits in that structure, assuming that all bits in that structure have the same circuit composition and, hence, the

same raw FIT rate. Then, the AVF of a hardware structure is equal to:

$$\frac{\textit{average number of ACE bits resident in a hardware structure in a cycle}}{\textit{total number of bits in the hardware structure}}$$

or,

$$\frac{\Sigma\ \textit{residency (in cycles) of all ACE bits in a structure}}{\textit{total number of bits in the hardware structure} \times \textit{total execution cycles}}$$

Another method is to use Little's Law:
N = B×L
where N = *average number of bits in a box*, B = *average bandwidth per cycle into the box*, and L = *average latency of an individual bit through the box*.

The AVF of the structure can then be expressed as:

$$\frac{B_{ace}\ \mathrm{X}\ L_{ace}}{\textit{total number of bits in the hardware structure}}$$

In this thesis, we use the 2^{nd} equation to compute AVF for the RFC. What needed are:

- sum of all residence cycles of all valid RFC entries during the execution of the program,
- total execution cycles for which we observe the ACE bits' residence time,
- total number of bits in the RFC (each entry is composed of 64 bits for a x86-64 family processor).

Experiments and Results

The aforementioned AVF is a relatively accurate metric to the designers than mere guardbaning. We would like to compute the AVF for the RFC architecture by incorporating the

MARSSx86 simulator with the ability to track the physical register assignment. In order to setup a baseline, we first ran a set of experiments in computing the AVF of a traditional monolithic 1-cycle register file (Figure 4.14). Since the access patterns of the integer register file and the floating-point register file differ in a integer-dominated and floating-point-dominated benchmark we separated AVF of these two register files.

We can observe that in both the integer (INT) and floating-point (FP) benchmark the AVF of the integer register file is generally higher than that of the floating-point register file. Though the AVF of the integer register file are both around 50%, the AVF of the floating-point one, however, is higher in the FP benchmark. The reason is apparent since in the FP benchmark the floating-point register file is heavily used and thus has more ACE bits than in the INT benchmark.

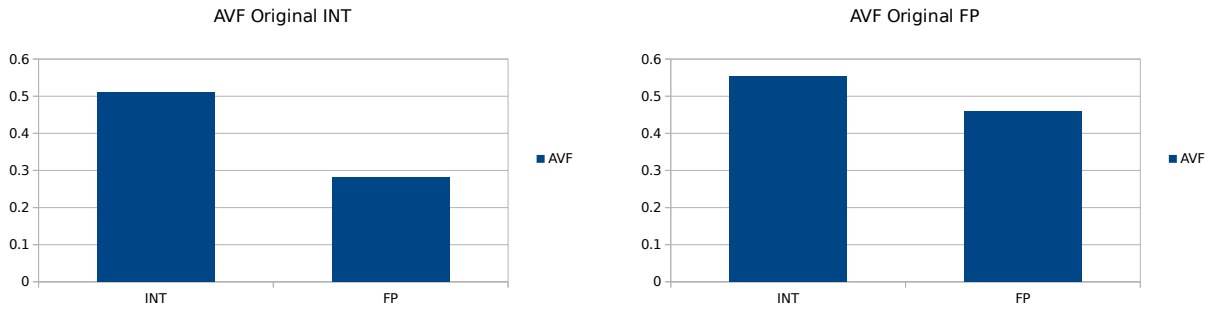

Fig. 4.14 The AVF of a monolithic 1-cycle register file

Figure 4.15 depicts the AVF of the RFC architecture in various sizes. Compare to the AVF of the monolithic 1-cycle register file, the AVF of the RFCs are generally higher (up to 80% in the INT benchmark and 90% in the FP benchmark for the integer register file). This is because unlike the monolithic register file, only the register values recently assigned are brought to the RFC which means the entries in RFCs are more probable to contain ACE bits. As a general trend when the size of the RFC becomes bigger the AVF decreases. One explanation is that as the size of the RFC increases the replacement (LRU) becomes less frequent which results in more un-ACE bits resides in the RFC.

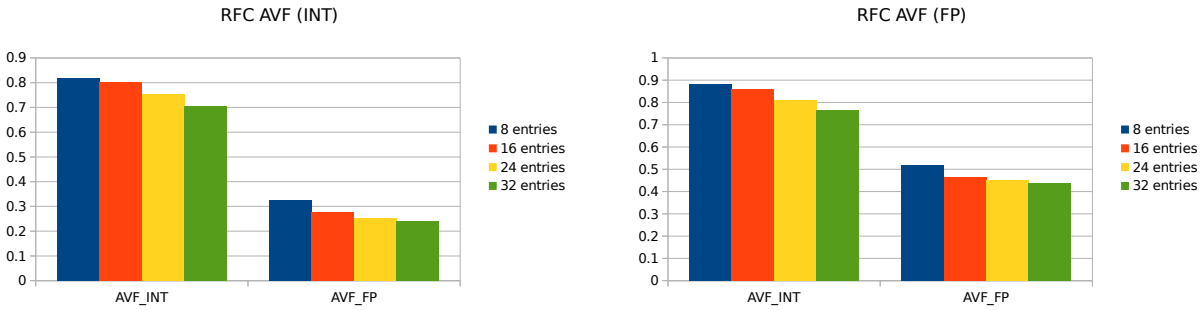

Fig. 4.15 The AVF of the RFCs

4.7.3 Soft Error Handling By The RFC

The register file cache architecture is essentially a partial redundant component of a traditional register file which can be used to handle the SEU. We would like to evaluate how the SEUs impose an influence on the RFC architecture. Our mechanism for the RFC architecture to handle SEU is the following:

If an SEU occurs, the affected μop is marked as "re-issue" and being sent back to the issue queue where it was once were. All the succeeding μops that have dependencies on that one will be hold in the issue queue as are normally handled while the faulty cell block will be marked as "invalid" and all the corresponding value will be read directly from the register file itself (which takes 2-cycles). The faulty cell block will be "fixed" when the value is flushed and the new value from the register file will be correct again.

For the simulation, apart from embedding this mechanism into the simulator we also needed to change the minimum number of buses from the previous 2 buses to 3 buses, because in extreme cases where all the 3 operands of one μop are hit by the soft error then these 3 operands needs to reach the RFC in the same time or a possible dead-lock might be created. To simulate the random nature of the soft error we modified the simulator to randomly inject soft errors in every certain cycles. As a baseline, Figure 4.16 presents the performance results of the 3-bus RFC. Figure 4.17 shows the performance of a 3-bus RFC with soft errors injected every 10 cycles.

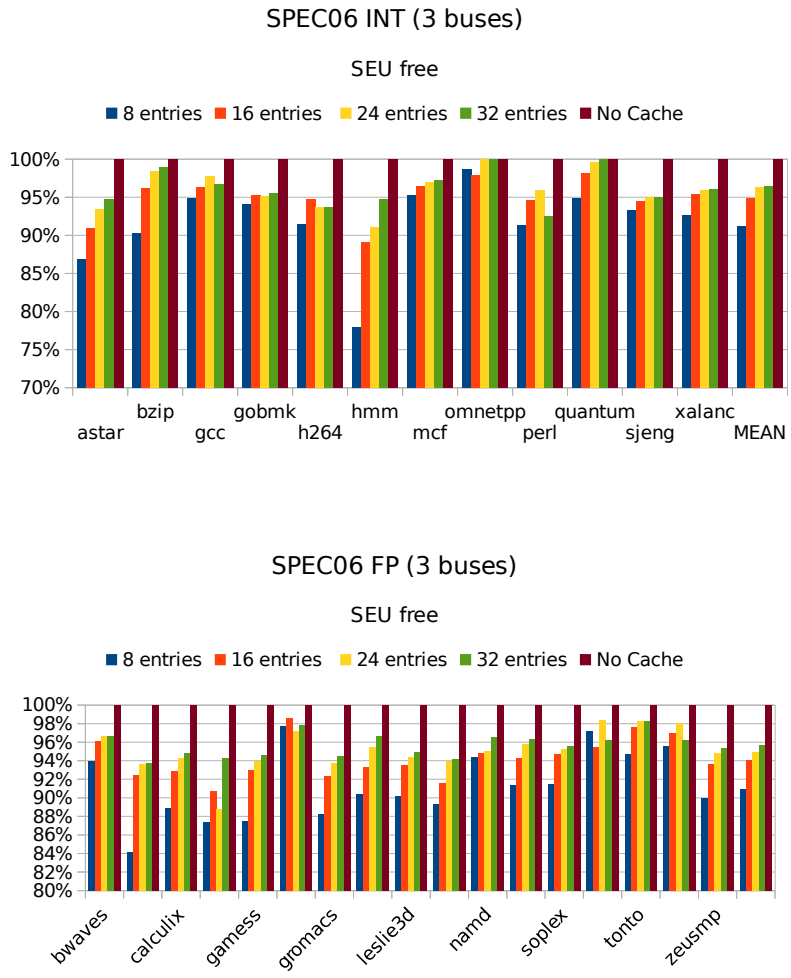

Fig. 4.16 Performance of 3 buses without SEUs

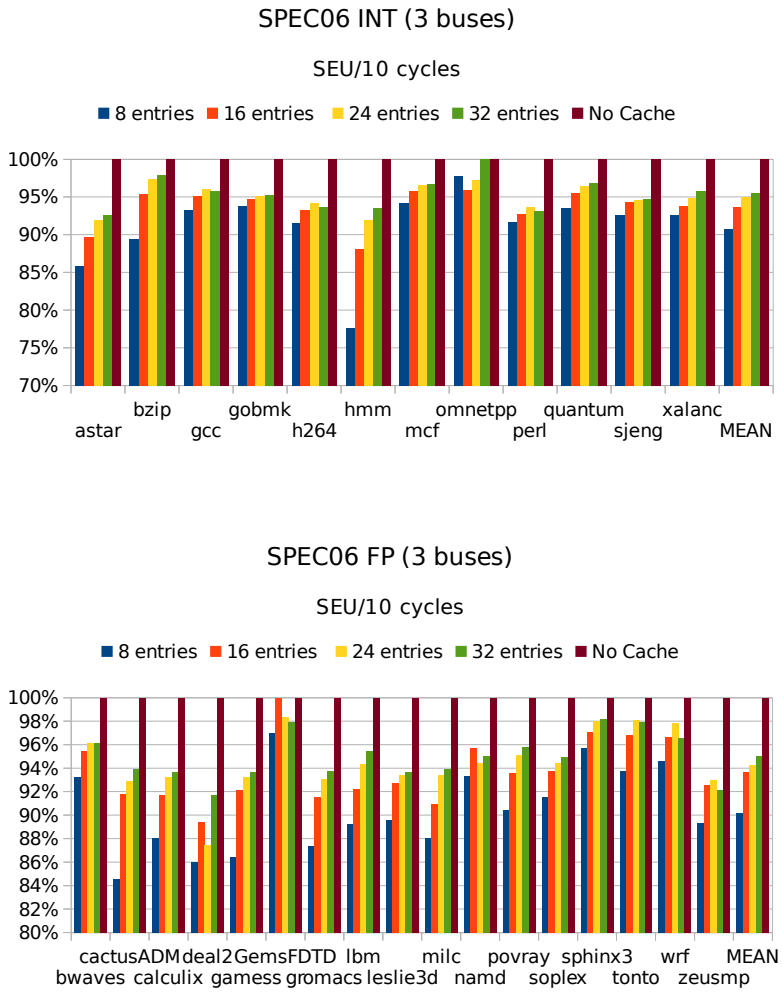

Fig. 4.17 Performance of 3 buses with SEUs

We can observe that the performance gap between Figure 4.16 and Figure 4.17 is minimal, less than 2%. With an injection rate of SEU/10 cycles the result is quite surprising. We conducted another set of experiments to find out the reason. As shown in Figure 4.18 when a cell block in the RFC is stricken (hit by an SEU) only less than 25% of such blocks were read again afterwards and most of them were not used again until being replace by a new and correct value.

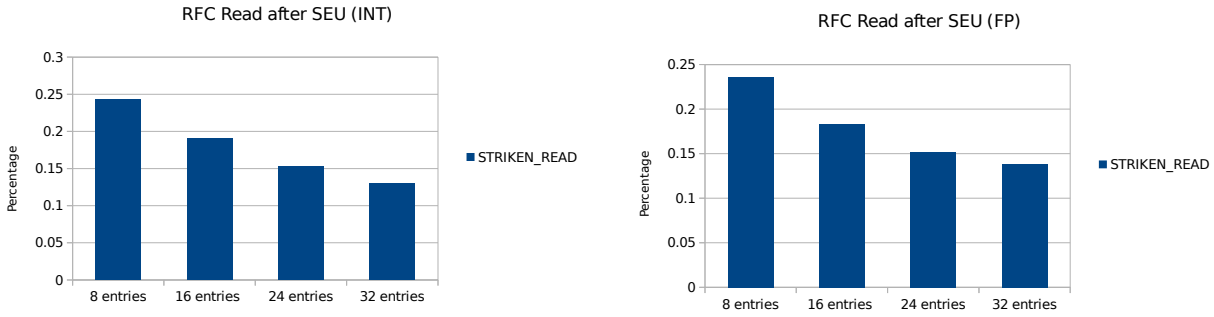

Fig. 4.18 RFC Read after SEUs

Chapter 5

Conclusion

In this thesis, we propose to use a cache-like organization of the register file in order to mitigate the performance drop caused by the process variation under current and future technologies or excessive use of the guard-banding. Since it is an architecture that has already been proposed before, we focus on using simulations to measure its performance and the influence of some of its key characteristics (register file cache size, buses in between the register file and the register file cache etc.).

We first observe that for a weak register file (which means that the size of the register file itself is shrinking over time) the register file cache architecture is able to retain most of its performance until the point in which the size of the register file is close to or even smaller than the physical registers of a given processor.

We then set out to find out some key features of this architecture and their relationships with the performance. The result shows that the size of the register file cache helps to prevent some of the performance loses to some point (in our case the register file cache size of 32). Meanwhile, the number of buses in-between the register file and the register file cache plays a minor role as far as performance is concerned. We go further to dig up the reason of this. Several experiments on bypass usage indicate that the bypass network on the process has been extensively used. We study the 2-operand μops, which is the type of majority of the μops, it turns out that these μops heavily use the bypass network which renders the buses in-between the register file and the register file cache idle in many cycles.

As an additional component on the processor the power consumption and the extra area are also a big concern. A register file cache architecture implies more transistors over a conventional register file and it inevitably increases the power consumption. Our study shows

that the area increasing is around 0.2% and is positively correlated with the register file cache size. On the other hand, the overall power consumption overhead is around 0.4%. Nevertheless, the positive correlation as seen in the area comparison breaks at the register file cache size of 32, it actually consumes less power than the register file cache size of 24.

We also added a soft-error handling mechanism to the RFC and try to find out the impact of the performance with the presence of soft error. We found that the RFC architecture is quite robust when facing off against the soft errors with only less than 2% of the performance degradation. The reason lies in the fact that most of the faulty cell block are not used any more after being stricken. The correct value then "flushes" the faulty value away.

Under the presence of process variations and the soft errors the register file cache architecture is shown to be a better choice than a 2-cycle register file using guard-banding bigger transistors in terms of performance. This architecture enables the use of some other techniques to further tackle the process variation problems. This is out of the scope of this thesis but it could open new doors to the research on process variation.

References

[1] A.Abramo, C.Fiegna, and F.Venturi (1995). Hot carrier effects in short mosfets at low applied voltages. *Proceedings of International Electron Device Meeting*.

[2] B.E.Deal, Sklar, M., Grove, A. S., and E.H.Snow (1967). Characteristics of the surface-state charge (qss) of thermally oxidized silicon. *J. Electrochem. Soc*.

[3] C.E.Blat, E.H.Nicollian, and E.H.Poindexter (1991). Mechanism of negative bias temperature instability. *Journal of Applied Physics*.

[4] Corporation, S. P. E. (2011). Spec cpu2006.

[5] Crupi, C.Pace, G.Cocorullo, G.Groeseneken, M.Aoulaiche, and Houssa, M. (2005). Positive bias temperature instability in nmosfets with ultra-thin hf-silicate gate dielectrics. *Journal of Microelectronic Engineering*.

[6] Cruz, J.-L., González, A., Valero, M., and Topham, N. P. (2000). Multiple-banked register file architectures. *International Symposium on Computer Architecture (ISCA'00)*.

[7] Gaillard, R. (2011). Soft errors in modern electronic systems. *Spriner*.

[8] Ghosh, S. and Roy, K. (2010). Parameter variation tolerance and error resiliency: New design paradigm for the nanoscale era. In *Proceedings of the IEEE*.

[9] Jaksic, Z. and Canal, R. (2013). Comparison of sram cells for 10-nm soi finfets under process and environmental variations. *IEEE Transactions on Electronic Devices*.

[10] L.Henning, J. (2006). Spec cpu2006 benchmark descriptions. *SIGARCH Computer Architecture News*, 32.

[11] Li, S., Ahn, J. H., Strong, R. D., Brockman, J. B., Tullsen, D. M., and Jouppi, N. P. (2009). Mcpat: An integrated power, area, and timing modeling framework for multicore and manycore architectures. In *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'42)*.

[12] M.A.Alam, B.Weir, and A.Silverman (2002). A future of function or failure. *IEEE Circuits Design Magazine*.

[13] M.F.Li, G.Chen, C.Shen, X.P.Wang, H.Y.Yu, Y.Yeo, and D.L.Kwong (2004). Dynamic bias-temperature instability in ultrathin sio2 and hfo2 metal-oxide semiconductor field effect transistors and its impact on device lifetime. *Journal of Applied Physics*.

[14] M.T.Quddus, T.A.DeMassa, and J.J.Sanchez (2000). Unified model for q(bd) prediction for thin gate oxide mos devices with constant voltage and current stress. *Proceedings of Microelectronic Engineering*.

[15] Mukherjee, S. S., Weaver, C., Emer, J., Reinhardt, S. K., and Austin, T. (2003). A systematic methodology to compute the architectural vulnerability factors for a high-performance microprocessor. In *Proceedings of the 36th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'36)*.

[16] Nair, A. A. and John, L. K. (2008). Simulation points for spec cpu 2006. *IEEE International Conference on Computer Design (ICCD)*, pages 397–403.

[17] Naseer, R., Bhatti, R. Z., and Draper, J. (2006). Analysis of soft error mitigation techniques for register files in ibm cu-08 90nm technology. *IEEE Transactions on Very Large Scale Integrated Circuits*.

[18] Patel, A., Afram, F., Chen, S., and Ghose, K. (2011). MARSSx86: A Full System Simulator for x86 CPUs. In *Design Automation Conference 2011 (DAC'11)*.

[19] Patil, H., Cohn, R., Charney, M., Kapoor, R., Sun, A., and Karunanidhi, A. (2004). Pinpointing representative portions of large intel®itanium®programs with dynamic instrumentation. *Proceedings of the 37th International Symposium on Microarchitecture (MICRO'37)*.

[20] P.Hamcha, T.Kamik, J.Maiz, S.Walstra, B.Bloechel, and J.Tschanz (2003). Neutron soft error rate measurements in a 90-nm cmos process and scaling trends in sram from 0.25-μm to 90-nm generation. *IEEE Transactions on Electronic Devices*.

[21] Sherwood, T., Perelman, E., Hamerly, G., and Calder, B. (2002). Automatically characterizing large scale program behavior. *Proceedings of the 10th international conference on Architectural support for programming languages and operating systems (ASPLOS X)*.

[22] S.Jafar, Y.H.Kim, V.Narayanan, C.Cabral, V.Paruchuri, B.Doris, J.Stathis, A.Callegari, and M.Chudzik (2006). A comparative study of nbti and pbti (charge trapping) in sio2/hfo2 stacks with fusi, tin, re gates. *Proceedings of Very Large Scale Integrated Circuits (VLSI)*.

[23] S.Sirichotiyakul, T.Edwards, C.Oh, R.Panda, and D.Blaauw (2002). Duet: An accurate leakage estimation and optimization tool for dual-vt circuits. *IEEE Transactions on Very Large Scale Integrated Circuits*.

[24] T.H.Ning, P.W.Cook, R.H.Dennard, C.M.Osburn, S.E.Schuster, and H.Yu (1979). 1 μm mosfet vlsi technology: Part iv-hot electron design constraints. *Transactions of Electron Devices*.